ERG2081 Final Report

Yao, He 118010095

Abstract— **This project builds a UAV-vehicle interaction platform using Unreal Engine 4 (UE4) and AirSim. The vehicle movement simulation environment is constructed in UE4 and the control of UAVs are realized using AirSim. Based on the platform, I conducted a series of experiments to analyze and verify the UAV's time and energy performance when it can ride on the vehicles. The demo can simulate the aerial-ground information exchanging and riding decision process of the UAVs.**

I. INTRODUCTION

The target of the project is to build a UAV-vehicle interaction platform so that further research can be carried out on this platform. The software we chose are Unreal Engine together with AirSim developed by Microsoft. Throughout the semester, I mastered the functional principle of the software and application development skills based on the software. A platform that is suitable to test our algorithms is designed and several experiments are carried out on the platform.

The major work consists of the following several parts:

1) Learn and understand the basic system architecture;
2) Learn how to use Unreal Engine and AirSim to create necessary environment and functions for our algorithms;
3) Build the simulation environment using UE4 and design testing scripts and libraries using AirSim;
4) Test algorithm on the platform: experiments on UAV's energy and travel time corresponding to different riding distance.

The following sections will follow the structure above, and discuss about details inside each section. At last, I will give some comments on the platform. Besides, while it worth pages to describe how to build and maintain the software on a computer, it is relatively less important for this report. Therefore, it is not covered here. For more details on this topic, I provided enough information on my previous reports, (report 1 and report 9), so future users can refer to these two reports as a manual.

In the rest of the report, Section II introduces the system architecture. Section III describes the simulation environment built in UE4. Section IV summarizes the APIs and library I developed for the NCEL UAV platform. I also listed my experience and several tips for the later developers on this platform. Section V presents our experiments and analysis. Finally, I conclude this project in Section VI.

II. SYSTEM ARCHITECTURE

The basic working frame consists of two components:

- **Unreal Engine (UE4)**
  It is the main simulation engine. This is the place where the simulation is designed and tested. There are two types of elements required for UE4 projects. The first one is called static meshes. They are static elements that make up the simulated scenary. For example, roads, buildings and light. The second type is called Actors, such as the AI cars I presented in one of my previous videos. They usually contain a variety of Static Meshes, and perform specific tasks according to scripts or programs. The major work regarding to UE4 is to build and use Static Meshes and Actors.
- **AirSim**
  Together with Pycharm and visual studio code, AirSim provides an algorithm testing and data gathering interface. UE4 alone cannot fully meet our needs since it does not offer UAV control and data collection functions. AirSim provides UAV models and control functions. It exposes APIs so users can interact with AirSim vehicles by sending real-time request to, and getting real-time data from UE4. Such powerful function is achieved because AirSim utilized *msgpack-rpc protocal* over TCP/IP throough *rpclib*. This property allows us to program outside UE4 to process the data and proceeding analysis without bothering complicate structures and policies of UE4. Thus, AirSim is the interface to test our algorithms and gather experimental data.

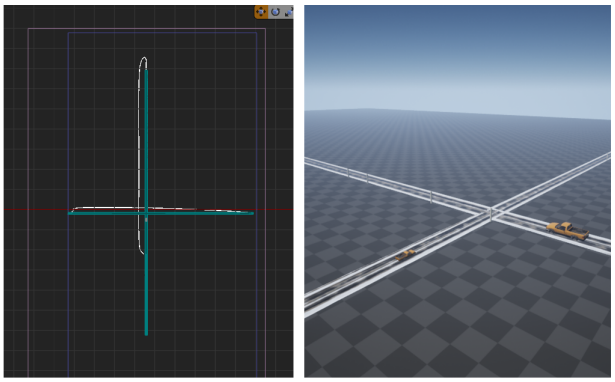


Fig. 1. The overview of the simulation world. On the left is the aerial view of the world. On the right is the real simulation world you will see when simulation begins. This world is mainly made up by two roads and two cars. Along each road, the white line is the *path*. Though it is a simple world, it contains all the necessary components for our simulation. We can add more objects to the world to make it the environment as close as to the reality.

III. Build simulation environment on UE4

Originally, UE4 is designed to make video games. An important property is that UE4 virtually creates an environment near-real to the physical world. Therefore, we use it to build the simulation environment that supports the demos and experiments in this project. As described in the previous section, there are mainly two types of elements: Static Meshes and Actors. Therefore, (1) using Static Meshes to build up the simulation world, and (2) designing AI cars using Actors are the major works when using UE4 to build the simulation.

A. Using Static Components to Build Up the Simulation World

Constructing Static Meshes is complicated, but luckily there are a lot of advanced projects and models contributed by other groups in the community. The models we use include: (1) a road model, which indicates the path vehicles travel on, and (2) a truck model. Based on a template provided by UE4, the roads and vehicle models are added to the simulation scene. Besides these two elements, I created and added extra Static Meshes including:

1) Objects called *marks*, that help identify locations in the simulations. These marks will also be used to construct a coordinate system in the testing scripts. Such coordinate system frees us from bothering the confusing coordinate system provided by UE4 and AirSim. Usually, the coordinates we computed are in the newly constructed system. The testing script will automatically transfer the coordinates to corresponding positions in UE4.

2) A target object that indicates the landing position of the drones.

3) Objects called *path*. In the simulation, roads help audience capture what routes the vehicles travel on, while inside the program, *path* helps the program identify the routes. I designed *path* using *spline* object provided by UE4. They are placed overlap with the roads. When AI cars start to drive automatically, they will follow *paths* to travel.

Fig. 1 shows the map of the created world and the overview of it. There are two roads perpendicular to each other. The white line along each road is the *path*. There is one car placed at one end of each road. When simulation begins, these cars will drive automatically. More objects can be added to the simulated world to make the world more fancy, but currently we have all the necessary objects for our simulation.

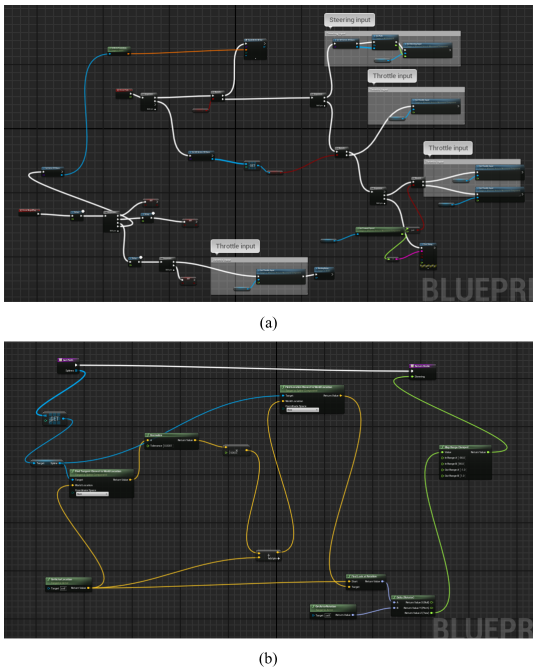
B. Designing AI Cars using Actors

Actors will perform specific tasks after simulation begins. Unlike Static Meshes that can be directly put into the world, we should program the tasks first. This is one major challenge when building the platform. The basic model for AI car is achieved at beginning of the semester, but the improvement on the control and stability lasted till the end of the project. UE4 provides a visualization programming interface. Using this interface and documentations from UE4 official website, I construct AI cars that can automatically drive along *path*. Fig. 2 shows the overview of the program. This program consists of the following components:

1) A function called *getPath()*, that gives the path information to the cars. As shown in Fig. 2.

2) Throttle and steering input, that allows vehicles to move and make turns.

3) A control function that stabilize the velocity of the vehicles. Currently, there are a lot of control algorithms to stabilize the speed of robots, such as PID control. However, that is not the focus of this project. Therefore, I simply used a feedback loop to control the car speed, so that it will oscillate around a certain value, making the



(a)

(b)

Fig. 2. The overview of AI car programs. (a) is the main program. When simulation starts, this program will be executed and AI car will drive automatically. (b) is the getPath() function. It will gives the path information to the cars, and thus control the direction of the cars.

speed approximately constant.

4) Keyboard instructions that controls the driving of cars if needed.

5) Variables that help program record and identify stages of the simulation.

With this program, the vehicles can drive along certain paths with a stable velocity. We can also use the keyboard instructions to control movements of the vehicles.

IV. Using AirSim and UE4 to Implement the Experiment

AirSim works as a plugin of UE4. Regarding AirSim, there are major three works: (1). Learning and understanding the principle of AirSim, (2). Learning AirSim APIs and using these APIs to design our experiments, (3). Learning how AirSim communicate with UE4, and designing new APIs for our work.

A. Principle of AirSim

AirSim exposes APIs so we can interact with the AirSim vehicles in the simulation programmatically. We can use these APIs to retrieve images, get states, control the vehicles and so on. The APIs are exposed through the RPC, and are accessible via a variety of languages, including C++ and Python. AirSim APIs use *msgpack-rpc protocal* over TCP/IP throough *rpclib*. When AirSim starts, it open port 41451 and listens for incoming request. Messages are sent to UE4 through this port. When the requests sent out, we are able to control vehicles, collect data, etc. The programming can be completed outside UE4, usually using Pycharm or Visual Studio Code.

B. AirSim APIs and Library for our lab: *NCEL_lib.py*

AirSim API library consists of three types of APIs:

1) **World APIs**
These APIs are for environmental setting and collecting data, etc. In terms of environmental setting, there are APIs enabling wind and fogs, listing objects inside the simulation world, and setting objects position and scales. In terms of collecting data, there are APIs collecting GPS and lidar data, collecting images, and recording the simulations.

2) **Drone APIs**
These APIs are only applicable when the simulation mode is drone mode. They controll the movement of drones. There are APIs controlling drones to take off and land. We can also control the drone fly to a specific position. The target coordinate can either be in the world frame, or in the frame refer to the drone itself. We can also control the rotation of the drones. Recently, the AirSim community designed new APIs that represent the rotor state of drones. These new APIs help us to compute the energy consumption of the drones.

3) **Vehicle APIs**.
These APIs are only applicable when the simulation mode is car mode. They will be used to control the movement of cars.

It seems that all these APIs are good enough for the simulation. However, one typical problem is that the coordinate system we use in the experiment is different from that in UE4 and AirSim. What's worse, there are a lot of bugs when using them. One common bug is that the task will not be executed completely.

To mitigate on these problems, I designed a library call *NCEL_lib.py* based on AirSim. This library features on the following:

- **A coordinate transformation system**
The library will create a new coordinate system for us, which is more intuitive and easy to analysis.

It frees us from the confusing coordinate system in UE4 and will automatically help us convert the coordinates between UE4 and our coordinate system.

- **Improvements on the AirSim APIs**
  For those improved APIs, a procedure to check whether the tasks are completed is implemented. The improved APIs will be executed until the tasks are completed. This ensure the stability and effeciency of our experiments. The imporved APIs can also be executed in a more easy and convenient scheme.
- **An algorithm that controls the landing of drones**
  In our library, I use a simple feedback loop to control the landing of drones. In the future, we can try more advanced and stable algorithms.
- **Functions computing the energy of drones**
  For the drone models in AirSim, the power of each rotor is given by (1)

$$P = \frac{K_v}{K_t}\tau\omega \tag{1}$$

  where P is the power, τ is the motor torque, ω is the angular velocity, K_v is the torque proportionality constant and $K_t = \frac{60}{2\pi K_v}$. A more simple approach to compute energy is utilizing the conservation of energy. Then (1) is equivalent to

$$P = Tv_h \tag{2}$$

  where T is the thrust and v_h the air velocity. All these parameters can be obtained from AirSim. Through (1) or (2) we can calculate the power consumed by each rotor, and finally compute the power consumed by the drone.
- **A function computing vehicles' speeds**
  Such API is not provided in AirSim, so this function requires implementing a new API, which is covered in the next section.

In the future, if someone is going to use UE4 and AirSim to run experiments in our lab, he/she can directly use this library, and make improvement on this library.

C. Implementing new APIs for AirSim

Implementing a new API is one core part of using AirSim and building the platform. In our demo, there is a decision process, which requires the speed of vehicles. However, there is no APIs getting vehicle speed or simulation time (which is required to compute speed) in AirSim, so we should implement this API.

AirSim API and Unreal Engine communicate through *rpclib*. According to the documentation of *rpclib*, there should be two interfaces to communicate through it:

- A server interface, that calls method *bind* to bind a string (function name) to a function.
- A client that calls method *call*. This *call* method will call the specific function using the specified function name and function parameters.

To add the RPC code to call the new API, A good strategy is following the implementation of other APIs defined in the files. A standard procedure is described below:

1) Add an RPC handler in the server which calls the implemented method in *RpcLibServerBase.cpp*. Vehicle-specific APIs are in their respective vehicle subfolder.

2) Add the C++ client API method in *RpcClientBase.cpp*.

3) Add the Python client API method in *client.py*. If needed, add or modify a structure definition in *types.py*.

Adding new APIs requires modifying the source code. Much of the changes are mechanical and required for various levels of abstractions that AirSim supports. In addition, when I did this project, I report the need for a guide of adding new APIs to AirSim community. The developers update the documentation for it a few days ago and future learners are recommended to check the link below: https://github.com/Microsoft/AirSim/commit/f0e83c29e7685e1021185e3c95bfdaffb6cb85dc.

V. Experiments on the Platform

This section describes experiments about UAV riding strategy carried out on the platform. We would like to test the optimal riding distance for the UAV to ride on a vehicle. Given that in reality vehicle and UAV cannot avoid uncertainty in their travel, such as speed variations, we also would like to see how much variations there are compared to our theoretical UAV riding performance results. The experimental scenario is a basic one, and it can be extended to include more number of cars and UAVs, together with environment variables such as wind and road condfitions and so on.

A. Experiment Procedure

Fig. 3 shows the map of the experiment. There are

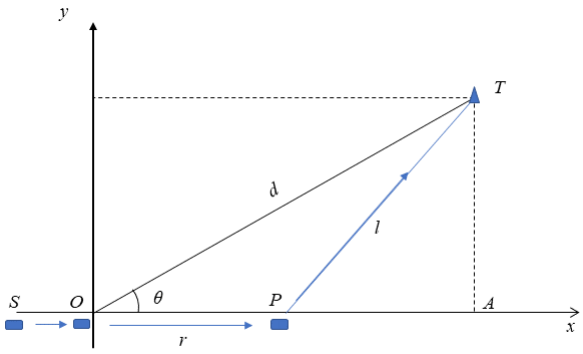


Fig. 3. The map of the experiment. O is the origin of the coordinate. T is the target position for the UAV. A is the projection of T onto x axis. P is some point between The blue rectangle is the car. S is the point where car starts to drive. When the car approaches to O, it will stop and UAV lands onto the car. Then the car start to drive. When the car pass P, the UAV takes off and flys to T.

three stages of the experiment:

1) Car starts to drive at S, and stops at O. This stage, the UAV will collect the speed of the car.
2) When car stops at O, the UAV flies and lands onto the car.
3) After UAV lands onto the car, the car drives along OA, carrying the UAV. Time starts to count.
4) At some point P between OA, the UAV takes off and flies to T along PT. The energy consumption starts to count.
5) UAV reaches T. Data collection stops, and experiment completes.

B. Theoretical Analysis

Denote the car speed as V_{car}, the UAV speed as V_{UAV}, $|OP| = r$, $|PT| = l$, and $|OT| = d$. Then l is given by

$$l = \sqrt{d^2 - 2dr\cos\theta + r^2} \tag{3}$$

The time spent on OP is t_1, and the time for UAV travelling from P to T is t_2. Then the total time for the UAV to arrive at T is $t = t_1 + t_2$. t can be computed as

$$t(r) = \frac{r}{v_{car}} + \frac{l}{v_{UAV}} \tag{4}$$

The energy consumption for the UAV is denoted as E. Using (2), the energy consumption is

$$E(r) = \int_{t_1}^{t} P(t)dt \tag{5}$$

We want to find an optimal r to achieve a good balance between energy consumption and time consumption. The cost function is given by

$$c(r) = \omega \times E_n(r) + (1-\omega) \times t(r) \tag{6}$$

where ω is the trade off parameter and $0 \leq \omega \leq 1$. $E_n(r)$ is the normalized energy consumption. Since the energy and time are measured in incomparable units, we normalized their units in studying their trade-off. $E_n(r)$ is given by

$$E_n(r) = \frac{E(r)}{\alpha} \tag{7}$$

where α indicates the unit energy consumption in each second, which is needed to be determined in the experiment. If the α is constant, then

$$E_n(r) = \frac{l}{V_{UAV}} \tag{8}$$

Minimizing the cost function, the optimal r is computed as

$$r^* = d\cos\theta - d\sin\theta \times \frac{(1-\omega)V_{UAV}}{\sqrt{V_{car}^2 - (1-\omega^2)V_{UAV}^2}} \tag{9}$$

C. Experiments

In the experiment, the target is located at point $T(470.9m, 836.5m)$, $d = 960m$, and the angle $\theta = 30°$. In this test, the car speed is kept at $5m/s$. The speed of the UAV is around $12.2m/s$. The speed might have small variation during the experiments. Running the simulations in the platform and gathering data from the experiments, the figures for total time, normalized energy consumption, and cost versus r are plotted in Fig. 4, Fig. 5, and Fig. 6, respectively.

For each r, the experiments are repeated three times. When computing the cost, the trade off parameter w is set to 0.8. Under such condition, the optimal value of take off position r^* is around $563m$. The standard deviations of recorded time, normalized energy and cost are 0.635, 0.583, and 0.588, respectively. During the

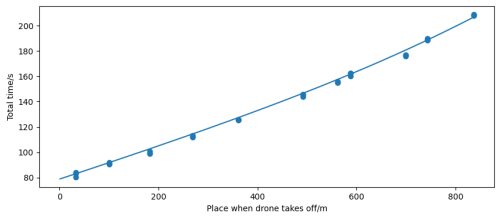


Fig. 4. Time/s elapses since drone lands onto the vehicle versus take off position y/m of the drone. The scatter data are the recorded data in the experiment. The straight line is the theoretical curve of it. The standard deviation of recorded time is 0.635.

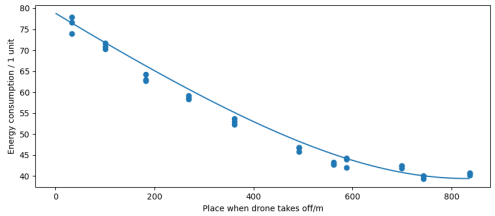


Fig. 5. Energy per unit required the drone flies to the target versus take off position y/m of the drone. The scatter data are the recorded data in the experiment. The straight line is the theoretical curve of it. The standard deviation of recorded energy is 0.583.

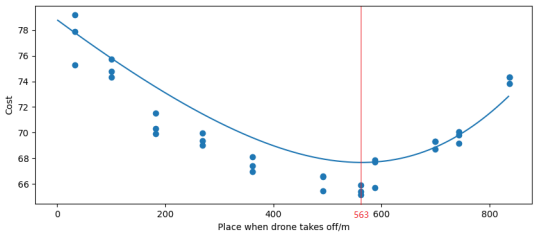


Fig. 6. Cost versus take off position y/m of the drone. The scatter data are the recorded data in the experiment. The straight line is the theoretical curve of it. The standard deviation of recorded cost is 0.588. The optimal value reported is $y^* = 563m$, which is highlighted in the figure.

experiment, the unit energy consumption α is around $63J/s$.

From the figures, the experimental data fit the theoretical curves. The experimental optimal cost is obtained around $r = 563m$. You may notice that the recorded cost may be lower than the theoretical value at some points. This might come from the variation of the vehicles' speeds in the testing. Further, there is an accelerating stage and decelerating stage during the flight. This will also affect the results. Regarding these effects, the experimental results are in consistence with analysis.

VI. CONCLUSIONS AND COMMENTS ON THE PLATFORM

In this project, the UAV platform is built based on UE4 and AirSim. I developed a simple simulation scenario and designed AI cars for the experiment. I also designed a library for the NCEL group for later developers who would work on this platform. Though the constructed world is simple, the modules and library created in the project cover most of the aspects we need for the experiments. The platform can be extended by many existing works, such as advanced UAV control algorithms, machine learning, and path planning algorithms.

I conducted experiments to learn and practice the UAV riding distance optimization. At the current setting, experiential results show that the optimal distance for the UAV to ride to closely matches our theoretical results, though vehicles and UAVs experience uncertainty due to acceleration/deceleration, taking-off and landing.

Current platform based on UE4 and AirSim has the potential to support most UAV-related experiments for researches, including computer vision, multi-agent systems, etc. However, since AirSim is still not fulling developed, a lot of functions remain to be explored and added by developers. Usually, we should implement APIs, modifying the code by ourselves, and dealing with all kinds of bugs during the researches. I would suggest later developers to keep in touch with the AirSim community, keep updating and make contribution.

REFERENCES

[1] AirSim documentations: https://www.zhihu.com/column/multiUAV
[2] AirSim repository: https://microsoft.github.io/AirSim/
[3] Shital, S., Debadeepta, D., Chris, L.. and Ashish, K., AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles, 2017, Field and Service Robotics, arXiv:1705.05065, https://arxiv.org/abs/1705.05065
[4] Unreal Engine 4 documentations: https://docs.unrealengine.com/en-US/index.html
[5] Nicola, Z., Multirotor Aircraft Dynamics, Simulation and Control, 2016, IEEE Computer Society, https://www.researchgate.net/publication/307513305